iCub connection documentation
Draft 1.00

Authors:
Giorgio Metta [pasa@liralab.it]

Last Update November 11th, 2006

1 Cable from the CAN bus (EsdCAN)

1.1 CAN connector: CAN-USB converter

The following table describes the connector that goes into the CAN-USB converter (EsdCAN):

Table 4: description of the connector to the CAN-USB converter.

PIN	Description	CAN cable
1	Trigger pin (not used)	
2	CAN low	Green
3	GND	Blue
4	No connect	
5	GND (shield)	
6	GND	
7	CAN high	Yellow
8	No connect	
9	V+	Red

NOTE: A terminator resistor (120 Ohm) connects “CAN low” and “CAN high”.

1.2 CAN cable

*** COLOR CODES TO BE VERIFIED ***

This paragraph describes the CAN bus **cable** (4 wires):

Table 5: description of CAN cable.

PIN	COLOR	Description
1	Red	CAN Vcc – Not used
2	Yellow	CAN high
3	Green	CAN low
4	Blue	GND

The Red & Blue wires are actually an external power supply (12VDC) but it is not needed with the ESD can (newer CAN-USB converter), this was required on the Value CAN device.

Notes:

- Termination resistor (120 Ohm) between pin 2 and 7 (CAN-HI, CAN-LOW)
- The red and blue wires are the bus power supply (not provided from the USB port): use 12V (range 6-24) from the motor power supply.

1.3 CAN connector: DSP

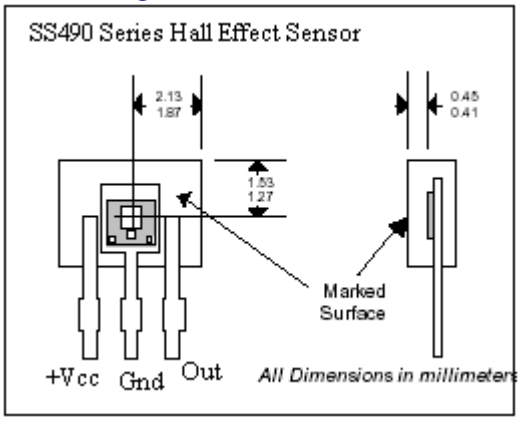
See details on the DSP manuals.

1.4 Additional information

Value Can: http://www.esd-electronics.com/
Connectors: http://www.rs-components.it (reseller)

2 Hall effect sensors

SS490 Series Miniature Ratiometric Linear Hall Effect Sensors (Honeywell, http://www.honeywell.com/sensing/).



Tipically, +Vcc (same as Vdd...)is a red cable, Gnd an orange one and Out (same as Signal...) a yellow one.

3 Power supply

*** TO BE COMPLETED ***

Currently in use: Xantrex XFR 35-35 1.2KW. See manual (for DC motors only).

3.1 Additional information: power supply

http://www.xantrex.com/

4 CAN bus protocol specification

The CAN bus is used at full speed 1Mbit (software must set parameters accordingly) and with standard messages (ID of 11 bits). See your card manufacturer manual for details of the API required for preparing the CAN bus adapter.

To allow each card in the bus to talk uniquely to each other card in the bus we divided the ID as follows:

3 bits	4 bits	4 bits
Message class	Source	Destination

The message class is fixed while the remaining bits can be subject to interpretation. Partitioning the ID according to this schema allows the hardware filtering of messages according to the specific goal of each card's control software.

Message classes are defined as:

Class #	Meaning
000	Polling messages
001	Periodic messages
010	Undef
011	Undef
100	Undef
101	Undef
110	Undef
111	CAN loader messages

Messages of undefined classes should not be received by any card.

For messages of class 000 the meaning of data/ID is defined as follows:

3 bits	4 bits	4 bits	Data[0]		Data[1-7]
Message class	Source	Destination	C	Message type	Payload

Where address 1111 is reserved for broadcast messages (see CAN loader specifications) and should be interpreted accordingly. "C" is the channel bit identifying the motor channel being addressed by the command contained in the "Message type" field. Message types are defined in "controller.h" in the controller_dc project.

Messages at the moment of writing are 74 (some undefined):

```
#define CAN_NO_MESSAGE                  0
#define CAN_CONTROLLER_RUN              1
#define CAN_CONTROLLER_IDLE             2
#define CAN_TOGGLE_VERBOSE              3
#define CAN_CALIBRATE_ENCODER           4
#define CAN_ENABLE_PWM_PAD              5
#define CAN_DISABLE_PWM_PAD             6
#define CAN_GET_CONTROL_MODE            7
#define CAN_MOTION_DONE                 8
#define CAN_WRITE_FLASH_MEM             10
```

```
#define CAN_READ_FLASH_MEM              11
#define CAN_GET_ENCODER_POSITION        20
#define CAN_SET_DESIRED_POSITION        21
#define CAN_GET_DESIRED_POSITION        22
#define CAN_SET_DESIRED_VELOCITY        23
#define CAN_GET_DESIRED_VELOCITY        24
#define CAN_SET_DESIRED_ACCELER         25
#define CAN_GET_DESIRED_ACCELER         26
#define CAN_SET_ENCODER_POSITION        29
#define CAN_GET_ENCODER_VELOCITY        61
#define CAN_SET_COMMAND_POSITION        62
#define CAN_POSITION_MOVE               27
#define CAN_VELOCITY_MOVE               28
#define CAN_SET_P_GAIN                  30
#define CAN_GET_P_GAIN                  31
#define CAN_SET_D_GAIN                  32
#define CAN_GET_D_GAIN                  33
#define CAN_SET_I_GAIN                  34
#define CAN_GET_I_GAIN                  35
#define CAN_SET_ILIM_GAIN               36
#define CAN_GET_ILIM_GAIN               37
#define CAN_SET_OFFSET                  38
#define CAN_GET_OFFSET                  39
#define CAN_SET_SCALE                   40
#define CAN_GET_SCALE                   41
#define CAN_SET_TLIM                    42
#define CAN_GET_TLIM                    43
#define CAN_SET_BOARD_ID                50
#define CAN_GET_BOARD_ID                51
#define CAN_GET_ERROR_STATUS            60
#define CAN_GET_PID_OUTPUT              63
#define CAN_GET_PID_ERROR               55
#define CAN_SET_MIN_POSITION            64
#define CAN_GET_MIN_POSITION            65
#define CAN_SET_MAX_POSITION            66
#define CAN_GET_MAX_POSITION            67
#define CAN_SET_MAX_VELOCITY            68
#define CAN_GET_MAX_VELOCITY            69
#define CAN_GET_ACTIVE_ENCODER_POSITION 70
#define CAN_SET_ACTIVE_ENCODER_POSITION 71
#define CAN_SET_CURRENT_LIMIT           72
#define CAN_SET_BCAST_POLICY            73
```

Message might or might not require a reply. A get message typically is sent by a host to a card to ask a specific piece of information which is returned using the same message

type. The ID of a return message is of the same class with the source and destination exchanged. The Data[0] is unchanged and the payload filled with the appropriate data.

For messages of class 001 the meaning of data/ID is defined as follows:

3 bits	4 bits	4 bits	Data[0-7]
Message class	Source	Message type	Payload

Where address 1111 is reserved for broadcast messages (see CAN loader specifications) and should be interpreted accordingly. Note that the destination address is here replaced by the message type which means that only 16 different messages are available in this modality. Messages of this class (001) are generated automatically by the card's software (according to a certain pre-configured policy) and sent on the CAN bus. They do not have a destination address since any card on the same bus should be able to read them. Interested cards can be listening on the bus for messages of a certain type originated by a specific card (4 bits are allocated for the source address as earlier). They typically contain status information. Data[0-7] is the payload of 8 bytes. Message types are defined in "controller.h" in the controller_dc project.

At the moment of writing the following messages are defined:

```
#define CAN_BCAST_NONE                  0
#define CAN_BCAST_POSITION              1
#define CAN_BCAST_VELOCITY              2
#define CAN_BCAST_FAULT                 3
#define CAN_BCAST_CURRENT               4
```

With the obvious meaning ☺.

For messages of class 111 the meaning of data/ID is defined as follows:

3 bits	4 bits	4 bits	Data[0-7]
Message class	Source	Destination	Payload

The message class of 111 is used to start the CAN loader activity and eventually upgrade the firmware of the DSP. The meaning of source and destination is clear, with the addition of the special address of 0xF (1111) which is reserved to specify a broadcast message (a common request to all cards).

Data[0] identifies the message according to the following list:

0xFF: request for board type and firmware version. The board replies with: 0xFF, board type of length 8bits (0 for motor control board, 1 for PIC/ADC board), version major number (8 bits), and version minor number (8 bits).

0x04: ping – checks that a certain card is alive, the card replies with 0x04 0x01. It can be used as a sanity check of the download procedure.

0x00: makes the firmware jump to the CAN loader code (the controller execution is effectively terminated). The loader section replies to this command appropriately by sending a 0x00, 0x01 back to the sender.

When the DSP is running in the download section CAN packets are parsed one by one. If nothing is received for N seconds then the loader jumps back to the main firmware section. While in loader mode the allowed messages are:

```
#define   CMD_BOARD      0
#define   CMD_ADDRESS    1
#define   CMD_START      2
#define   CMD_DATA       3
#define   CMD_END        4
#define   CMD_ERR        5
#define   CMD_BROADCAST 0xFF
```

We have already described the CMD_BOARD command (0x00) above.
The other messages are treated as follows:

0x01: receives the address, type and size of the next S-file packet. Data[1] is the length, Data[2] and Data[3] are the address, and Data[4] is the block type (program 0 or data 1).

0x02: an indication to start the program. It replies with 0x02, 0x01.

0x03: a data packet – something to be flashed – 6 bytes of payload are flashed to memory and a reply 0x02, 0x01 is sent back to the sender.

0x04: exits the main loop, the program is terminated. It replies with 0x04, 0x01.

0xFF: a broadcast message is receive. This is interpreted exactly as the request for board type, firmware version, etc. described before.

4.1 Additional information

Please see the controller code from the iCub CVS repository.